Identifying individual level patient data within algorithms and code prior to egress from Trusted Research Environments: worked examples using synthetic data.

M. Baragilly[1], A. Topham[2], S. Gallier[3], E. Sapey[4]

1. Dr Mohammed Baragilly.

   A. Institute of Inflammation & Ageing, College of Medicine and Health, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK. ORCiD ID: https://orcid.org/0000-0003-4775-4742 m.h.h.baragilly@bham.ac.uk
   B. Assistant Professor, Department of Mathematics, Insurance and Applied Statistics, Helwan University, Egypt.

2. Dr Alexander Topham, PIONEER: Health Data Research UK (HDRUK) Health Data Research Hub for Acute Care, University Hospitals Birmingham NHS Foundation Trust, Edgbaston, Birmingham, B15 2GW, UK. Alexander.Topham@uhb.nhs.uk

3. Dr Suzy Gallier (Corresponding Author).

   A. Technical Director of PIONEER: Health Data Research UK (HDRUK) Health Data Research Hub for Acute Care, University Hospitals Birmingham NHS Foundation Trust, Edgbaston, Birmingham, B15 2GW, UK. https://orcid.org/ 0000-0003-1026-4125. Suzy.gallier@uhb.nhs.uk.

   B. Head of Research Informatics, University Hospitals Birmingham NHS Foundation Trust, Birmingham, B15 2GW.

   C. Assistant Professor, Institute of Inflammation & Ageing, College of Medicine and Health, University of Birmingham, Edgbaston, Birmingham, B15 2TT.

4. Professor Elizabeth Sapey.

   A. Director of PIONEER: Health Data Research UK (HDRUK) Health Data Research Hub for Acute Care, University Hospitals Birmingham NHS Foundation Trust, Edgbaston, Birmingham, B15 2GW, UK. ORCiD ID: 0000-0003-3454-5482. E.Sapey@bham.ac.uk

   B. NIHR West Midlands Applied Research Centre, University of Birmingham, B15 2TT, UK

   C. NIHR Midlands Patient Safety Research Collaboration, University of Birmingham, B15 2TT

   D. NIHR Birmingham Biomedical Research Centre, University Hospitals Birmingham NHS Foundation Trust, Edgbaston, Birmingham, B15 2GW.

Key words: Secure Data Environments (SDEs), Synthetic Data, Data Egress, Privacy-Preserving, Data security

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Evidence before this study

Data security in health and social care research has increasingly focused on the use of Trusted Research Environments (TREs) to protect sensitive information, yet the risks of data egress from these environments remain underexplored. To assess the existing body of evidence, we conducted a PubMed search on September 03, 2024, using the terms “Trusted Research Environments” OR “Secure data environments” AND “disclosure” OR “Data egress” OR “Data breach” AND “health data security,” without language restrictions and limited to publications from 2014 to the present. This search yielded 287 results, only 1 publication directly addressed the risks of data egress from TREs, and even fewer provided empirical insights or worked examples relevant to health data and the risks of disclosure.

Our review identified that most literature focused on general privacy frameworks or the theoretical risks associated with data access rather than egress, highlighting a notable gap in practical, evidence-based resources. No studies were found that demonstrated specific methods by which data could be unintentionally egressed via code or machine learning outputs within a TRE. Consequently, to our knowledge, no published study has provided real-world examples of data egress risks in health research TREs, nor has any research systematically examined the effectiveness of current egress screening protocols in identifying potential data leakage. Notably, no systematic reviews or meta-analyses exist to quantify the risk of data egress from TREs, which remains a key gap that this study seeks to address.

Added value of this study

This study provides empirical evidence through worked examples, demonstrating specific scenarios where individual-level patient data could be unintentionally egressed from TREs within machine learning and data science outputs. Using synthetic datasets, we simulated three scenarios involving binary serialisation of data, complex objects, and plain text mark-up reports. Our findings reveal that current automated egress screening processes may overlook embedded data within these formats, underscoring the limitations of existing tools in managing egress risks in TREs. This study uniquely addresses the need for practical, hands-on insights into how such risks manifest in real-world settings, offering specific guidance for data controllers on mitigating these risks.

Implications of all the available evidence

The findings of this study have significant implications for policy and practice, especially within the context of NHS England’s ongoing transition to a model where health and social care data are accessed exclusively through SDEs. Our work demonstrates the need for robust manual egress review processes

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

alongside automated tools, suggesting that TRE and SDE controllers should prioritise training and ongoing monitoring of egress protocols. For future research, these findings prompt a need to develop more advanced, adaptable egress screening systems capable of detecting hidden data within complex code outputs. This study provides a foundation for further exploration into safeguarding data within TREs, helping align practice with the growing demand for secure, responsible data access in healthcare research.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Abstract

Background: Access to sensitive data for research has transitioned from a data egress model to the use of Trusted Research Environments (TREs), sometimes termed Secure Data Environments (SDEs), with only analytical code or aggregated data being exported. Theoretical risks of unintended data egress within code have been highlighted, but most TRE processes assess the risk of data access and not data egress. This paper provides worked examples where code / model egress from TREs can include patient level data.

Methods: Synthetic datasets with demographics and NEWS2 vital signs were created using SciPy and NumPy for two fictitious populations. These datasets were transformed for machine learning and embedded into various formats to simulate potential egress scenarios. Three worked examples include binary serialisation of data, binary serialisation of complex objects and plain text mark-up reports.

Findings: In each example, file size alone was insufficient to identify the presence of (here synthetic) individual patient level data. Furthermore, the programs being used did not issue any warnings about embedded data. Converting commonly used binary formats into more interpretable forms was only effective for code libraries that the automation explicitly recognised and requires maintenance to keep pace with updates to the target libraries. IN each case, manual inspection alongside automated processes proved the only means to identify and remove embedded data.

Interpretation: The cases highlight the complexities in identifying and preventing data egress from TREs. Key insights include clear guidance for researchers, recognising the limitations of binary serialisation for egress due to security vulnerabilities and screening complexity and the importance of plain-text data exports for ease of verification.

Funding: This work was funded by Health Data Research UK Medicines in Acute and Chronic Care Driver Programme.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Introduction

Increasingly, there is an expectation for health and social care data to be accessed for research in Trusted Research Environments (TREs), otherwise referred to as Secure Data Environments (SDEs).[1, 2] These environments enable researchers to access data for analysis, overseen through physical systems and processes and contractual arrangements such as data licensing agreements. Data remains within the TREs, eliminating the need to egress patient-level data to external researcher-controlled environments. This system aligns with the "five-safes" framework for privacy and providing greater reassurances to patients, the public and data controllers about the safety and security of accessing data for research.[3, 4]

Several initiatives have established TREs for to support specific research purposes. Examples include the Health Data Research UK (HDR UK) founded hub, PIONEER focusing on Acute Care[5], the UK Longitudinal Linkage Collaboration (UK LLC) [6] focusing on COVID-19 records, and the Safe Haven Artificial Platform (SHAIP) [7] which provides a secure environment for AI algorithm development using anonymised NHS imaging and reports. At a national level, the Secure Anonymised Information Linkage (SAIL) Databank [8] serves as the TRE for Welsh NHS and social care data. Recently, NHS England (NHSE) funded 12 regional SDEs to facilitate access to linked regional health data.[9] The policy surrounding this initiative emphasises a shift towards a model where all NHS data is accessed through NHS-controlled SDEs.

The SDE model operates under the assumption that the risk egressing potentially identifiable patient data (either by mistake or maleficence) is low, given egress permissions are tightly controlled. However, managing SDEs has become increasingly complex, particularly with the advent of data driven methodologies, such as machine learning (ML).

There is little published guidance on managing data egress from Trusted Research Environments (TREs), which remains a key risk for potential data breaches.[10] The GRAIMATTER project highlighted potential risks of ML models inadvertently exposing health data during output, posing reidentification risks and classifying outputs as personal data.[11, 12] However, while these theoretical risks have been considered, there are no worked up, practical examples to demonstrate this risk, limiting the sector's ability to mitigate effectively.

The overarching aim of this work was to support SDE data controllers to identify potential cases where data egress could occur. We provide worked examples of where individual patient level data could be egressed from SDEs within analytical outputs, without this being obviously apparent to SDE data controllers. The chosen examples are based on cases where

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

data egress was prevented and individual level patient data removed prior to outputs being shared. For this study, examples have been replicated using synthetic data. Sample files are provided for reference in the online supplementary material, illustrating how the data concerned might look within analytical code or algorithms.

Methods

Synthetic data generation

All examples used the same dataset, namely a set of demographics and NEWS2 physiological measurements for two fictitious populations of one hundred patients each, generated at random by drawing from a reasonable distribution of values. Data were generated using the random numerical distributions in the SciPy package in the scipy.stats namespace, and categorical variables were drawn using NumPy's random.choice method, using the parameters in Table 1.[13, 14] Code to generate the sample data is in Online Supplementary Table S1, including seed values for all pseudorandom number function calls, and the dataset itself is in Online Supplementary Table S1a. Ethnicity values are from the Office for National Statistics.

To generate further example files for the case studies, the randomly generated data in Online supplementary Table S2a was first feature-transformed into a dataset compatible with Machine Learning methods (Online supplementary Table S2b) using code as shown in Online supplementary Table S2c), and finally example files were generated using models trained in Supplementary file S2d.

<Insert Table 1 about here>

Table 1: Parameters used to generate random data for file samples presented herein. Pulse rate was selected to be an uninformative variable, and so parameters were the same for both populations. Age values were all whole integers ranging from 18 to 95. Ethnic groups were selected by equal probability, samples from the Office for National Statistics high-level ethnic categories for the 2021 census.[15]

To generate the code with extractable, embedded patient level data 100 randomly generated sets of synthetic patient data were developed containing demographics, outward code (first half of postcode), and some inferred sensitive diagnoses along with times of admission and discharge. This was then serialised to JSON and pasted into a copy of the Jupyter Notebook (.ipynb) used to generate the random data above at line 173 using a plain text editor (.ipynb files are themselves JSON formatted data). It was then confirmed that Visual Studio Code (VS Code) opened the file and displayed the notebook content without the fictional patient data.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

All development work was performed in VS Code version 1·89·1 with the Python extension version v2024·6·0, Jupyter extension version v2024·4·0 and Pylance version v2024·5·1, using Python version 3·10·5.

The risk caused by the analytical techniques described in the case studies was rated from low to high depending on the visibility of the patient level data, with low risk being defined as anything where sensitive data are explicitly identifiable and easily inspected, and high risk being defined as instances where sensitive data is concealed in unusual and/or technically demanding file formats.

Results

Case one: Binary Serialisation of Data

This case involves a request to egress performance data from an artificial intelligence generated algorithm trained on patient-level data. The use of Python's binary object serialisation ("pickle") to save the performance data matrices serialises other objects from memory to disk, including the matrices used to train the models. This data matrix includes patient level data. Online supplementary file S4 contains a worked example of a binary-serialised dataset which includes patient level data. Figure 1 provides a series of marked screenshots demonstrating the embedded synthetic data within the code.

<Insert Figure 1 about here>

Figure 1: Case one as viewed in VS Code. Screenshot taken from VS Code of Case one example output file, showing the object type name and the contents of the object highlighted in red. In this instance, the default display rendering of the object presented the data in a clear and very readable way, and the object type immediately reveals the nature of the object if not the content.

In this instance the data were easily identifiable by the type, although the technician would have to be cautious of namespace confusion attacks and inspect the actual contents of the object. This object (a Data Frame from the Polars python package) has a default display format that is easy to interpret.

This case highlights two challenges: first, even when researchers are aware of permissible egress data, tools like object serialisation or disorganised file management can easily result in including row-level patient data among outputs. Second, binary object serialisation complicates output verification. Even plain-text serialisation like JSON or XML requires some expertise to interpret.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

In this instance, the “pickled” files were read back into Python, which throws informative exceptions when dependencies are missing, allowing researchers to open them for inspection. The object’s type indicated its likely contents, but verifying its safety still required manual inspection, as a namespace confusion attack could circumvent this process.

Learning

File size alone did not provide a clear indication of the presence of patient-level data, as serialised binary files can vary in size. Additionally, the Polars Python package did not generate any explicit warnings or alerts to indicate that patient-level data might be embedded within the serialised object. The lack of an in-built warning mechanism or automated detection capability emphasises a gap in current methodologies for verifying serialised objects, especially when faced with sophisticated techniques such as namespace confusion attacks.

Robust monitoring strategies might include prohibiting the export of binary-serialised files where possible, to prevent the inadvertent exfiltration of patient data. While partial automation is feasible by developing tools that convert commonly used binary formats into more interpretable forms like .csv, this approach has limitations. It is only effective for code libraries that the automation explicitly recognises and requires continual maintenance to keep pace with updates to the target libraries. Thus, a combination of improved automated solutions, package-specific warnings, and manual review is necessary to mitigate the risk of data breaches.

Case two: Binary Serialisation of Complex, High-Risk Objects

Training AI models employing Python’s binary object serialisation can lead to patient data being embedded several layers deep in various object properties. A list containing objects was serialised, including a dictionary with AI models and their performance data. While the performance data was safe, a Support Vector Machine (SVM) included samples of the original training data in its support vectors, an acknowledged issue with SVM models.[16] The training data had been Z-transformed, which could be reversed using population aggregates requested for egress.

The SVM classifier was small in terms of kilobytes (kB) size but contained patient-level data in its support vectors. Online Supplement S5a contains a worked example of this case which includes synthetic patient level data in a SVM binary object in a complex, multilayered binary-serialised object, while Online Supplementary table S5c contains the support vectors in tabular format, extracted from the SVM. Online Supplementary file S5b contains steps a technician

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

might take to investigate the binary file, with the last step identifying the contents of the SVM in the screenshot of Figure 2 below.

<Insert Figure 2 about here>

Figure 2: Case two as viewed in VS Code. Screenshot of VS Code showing fragment of Supplementary Code file S05b wherein the presence of support vectors was confirmed for a support vector machine classifier object embedded in Online Supplementary file S05a. Highlighted in red are the points where the suspect property's presence in the object was confirmed, and a rendering of the property's contents.

The embedded data is identifiable with prior knowledge of potential safety risks associated with certain models, in this case the support vector machine. Here, the file size would probably be large but the SVM itself not necessarily so. Manual checking through multiple layers: list, dictionary, SVM, and support vectors, verifying contents by reversing the Z-transformation identified the data in question; see also Table 2 below for a full list of the properties of this object.

In hypothetical scenarios, a malicious actor could replace object properties with data to be exfiltrated. For instance, ndarray types in the Support Vector Classifier can be replaced with arbitrary data, bypassing detection unless directly examined.

Learning

File size alone did not reliably indicate the presence of patient-level data, as the SVM model itself was small in terms of kilobytes, despite containing sensitive information within its support vectors. Additionally, the binary object serialisation technique employed in Python did not trigger any automated warnings or alerts regarding the presence of embedded patient data. It was only through thorough manual inspection and domain expertise that the potential risk was identified, particularly given the known issue of SVM models retaining samples of the training data. The ability to identify such data requires a strong understanding of both the AI models involved and potential data safety risks.

Binary serialisation, while convenient, introduces risks such as compatibility issues with library updates, security concerns (e.g., remote code execution vulnerabilities), and difficulties in detecting manipulated or exfiltrated data. The risks are not confined to Python or R but extend to any platform where text-based serialisation might be manipulated to evade normal detection procedures.

<Insert Table 2 about here>

Table 2: Instance properties of a support vector machine classifier trained using the popular SciKit-Learn Python package ("SVC" class defined in the sklearn.svm namespace). Highlighted in red is the property that contained a large subset of the patient-level training data identified in Case three. Note also that in Python any one of the properties of a class object can be assigned at-will by external code, and so it is trivial to assign any arbitrary numpy.ndarray object (a data matrix of fixed type but arbitrary dimensions) to any of the relevant properties of a

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

class that normally contains them – the name and normal function of the property are absolutely not reliable indicators as to the content of the property in any given instance when egressing data.

Case three –Plain-Text Markup Reports

This case's main report was formatted using an R library that encodes data, code, and reports as an HTML file, embedding spreadsheets of underlying data in Excel format using Base-64 encoded strings of the .xlsx files, which are themselves zip-compressed XML files.[17]

The risk in this case includes the egress of aggregate numbers where low patient counts (sometimes equal to one) are unsuppressed. A malicious actor or simply a third-party library lacking robust maintenance of its file outputs could embed data in HTML files as comments or as an unused variable in JavaScript, making detection challenging.

Online supplement S3 contains a worked example which includes patient level data embedded in a Jupyter Notebook file, a commonly used type of JSON-encoded markup document. Figure 3 provides a series of marked screen shots demonstrating the embedded synthetic data within the Jupyter notebook file.

<Insert Figure 3 about here>

Figure 3: Patient data embedded in a Jupyter Notebook. Partial screenshots of the code markup file in supplementary file S03. A shows a screenshot of the end of the file as rendered in VS Code at the location into which the synthetic patient data was inserted, and B shows the markup for the same screen area displayed in a plain text editor (Notepad++), marked in red is the site in which synthetic patient data has been inserted.

In this instance the embedded data is detected by manual inspection of the markup document without rendering, in a plain text editor. A further complication in Case three is the embedding of file data in the markup document as Base-64 encoded text, which required specialised knowledge to identify and verify and would not be achievable by non-technical users.

Learning

Once again, the file size did not serve as an immediate indicator of the presence of embedded patient-level data, as markup documents like HTML can vary in size depending on their content and verbosity. Similarly, the program did not issue any warnings about embedded data, as the use of Base-64 encoding to embed Excel files within the HTML file, a common practice, went undetected. It was through manual inspection using a plain-text editor that the embedded data was identified.

This case demonstrates that while tools can generate comprehensive, interactive reports using markup languages like HTML, they introduce significant challenges for verification, particularly when dealing with embedded data or Base-64 encoding, which requires specialised knowledge to detect and interpret.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

There are risks associated with embedding patient-level data in images or metadata within these markup documents, as well as with third-party libraries that might not robustly manage file outputs, potentially allowing hidden data to be included without detection. To mitigate these risks, a more secure and efficient solution is to export simple, plain-text aggregate data with suppressed small numbers, facilitating safe reporting and formatting outside the SDE environment. This method minimises the risk of unintentional data breaches and allows for continued access to formatted reports without exposing sensitive data. Table 3 provides a summary of the cases.

<Insert Table 3 about here>

Table 3: Summary of all cases discussed herein, including the description, data involved, method of export, challenges faced, insights gained, and guidelines for practitioners to mitigate similar risks.

Discussion

Accessing sensitive data such as health data using SDEs /TREs is becoming commonplace. In the UK, there has been significant government investment to establish SDEs both at a regional and national level, accompanied by a stated policy for NHS health data to only be accessed through NHS-controlled SDEs.[2] Operating standards and frameworks are currently being agreed which mainly focus on data access processes. The focus on data access in SDEs reflects a commonly held belief that the egress of code from the SDE is low risk, as individual patient level data cannot be exported without this being explicitly clear due to file name, type, or size.

The examples included in the current paper build on an earlier publication which explored the theoretical risks of potentially identifiable patient data being egressed either accidently or maliciously from TREs.[18] The current paper goes further and demonstrates these risks with worked examples using synthetic data.

The purpose of these worked examples is threefold. First, to demonstrate that the theoretical risks highlighted in previous work represents a real risk of a data breach. Data Controller and organisations seeking to define standards for SDEs/TREs should consider processes for data egress as well as data access. Second, to provide some clear and specific examples of how individual level patient data can be obfuscated within code or files, so that Data Controllers can work through these examples and consider the required processes to identify embedded data and ensure it is removed prior to egress. The paper also provides potential strategies for each case study, to help Data Controllers avoid potential data breaches. This is not an exhaustive set of case studies, but they do highlight the types of areas, programmes and analytical approaches for data controllers to consider. Third, to make researchers aware of

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

these risks when they are developing algorithms or training them with a TRE/SDE, so that researchers themselves can avoid these practices. There is an increasing interest in using routinely collected health data for research, including complex statistical modelling and machine learning. Researchers will develop their own coding practices but sometimes the "short cuts" highlighted above can have the unintended consequences of embedding patient level data in the code. Understanding this risk provides an opportunity to address it. In keeping with this, Jefferson et al.[11] proposed practical recommendations grouped by project stages, emphasising collaborative strategy development.

In all cases presented, the hazard presented was full exfiltration of patient-level data, illustrating some examples where accidental data egress could occur, and the processes which can be used to mitigate this. Figure 4 provides a summary of the cases and the risks identified, as well as mitigating principles and practices.

<Insert Figure 4 about here>

Figure 4: Spectrum of risk for files egressed from the PIONEER TRE, the scale of which depends on the type of file proposed for egress. Plain-text is fully human-readable, but depending on the way it is used can be relatively complex in nature, but always explicit. Non-human-readable formats, typically binary objects derived from arbitrary, user-defined types present the greatest risk.

Within the PIONEER TRE screening prior to egress is performed manually by exploring research outputs and inspecting contents and metadata on a case-by-case basis. This covers a wide range of possible file formats, data structures, and content, but generally entails several common steps:

1. Validation of file content as expected from file extensions, e.g. that a file with the extension .jpeg is, indeed a JPEG image.
2. Assessment of the risk of a file containing embedded data structures that may hold sensitive data, e.g. .csv file contents are readily verified, while complex binary structures are much harder to verify the safety of.
3. Inspection of file contents, which can range from simple visual inspection in the case of small text-based files such as .csv or code files, to recursive inspection of the properties of serialised objects.
4. Presentation and sign-off by senior personnel.

This manual work matches reports for other environments such as in Australian law enforcement's "Data Airlock" secure environment.[19] Fully manual inspection has challenges with regards to consistency and timeliness and requires technical expertise. Automated tooling may support aspects of this flow, but this still requires manual inspection as programmes can flag that an object contains a large matrix of data but not whether that data is sensitive or not.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Conclusion

This paper presents examples to help Data Controllers mitigate the risk of data breaches during model exportation/egress from TREs/SDEs. To effectively tackle these challenges, this paper offers practical suggestions and guidelines for practitioners. These include implementing data leakage protection systems, investing in education and training programs, ensuring anonymity of sensitive data, and setting up serialisation safeguards. However, the main recommendation includes ensuring domain expertise of the programmes being used and manual inspection of egress files, until a more robust automated system is developed. By adopting these proactive measures, practitioners can significantly enhance the security of TREs, promoting trust, and fostering responsible research practices.

Data Sharing Agreement

To facilitate knowledge in this area, the synthetic data, analytical code and a data dictionary defining each field will be available to others through application to PIONEER via the corresponding author.

Author contribution

M. Baragilly, S. Gallier, and E. Sapey designed the study, collated data, performed some analysis, and wrote the manuscript. A. Topham performed and supported statistical analyses. M. Baragilly and S. Gallier wrote the first draft of the manuscript. All authors amended the manuscript and approved the final version.

Acknowledgements

This work was supported by PIONEER, the Health Data Research Hub in acute care, NIHR Midlands Patient Safety Research Collaboration (PSRC) and the NIHR Applied Research Collaboration (ARC) West Midlands. This work uses synthetic data, however this was generated based on data provided by patients and collected by the NHS as part of their care and support. We would like to acknowledge the contribution of all staff, key workers, patients and the community who have supported our hospitals and the wider NHS. This work was funded by Health Data Research UK Medicines in Acute and Chronic Care Driver Programme. This research is also funded by the National Institute for Health and Care Research (NIHR) Midlands Patient Safety Research Collaboration (PSRC) and Birmingham Biomedical Research Centre (BRC). The views expressed are those of the author(s) and not necessarily those of the NIHR or the Department of Health and Social Care.

Conflicts of Interest

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

M Baragilly reports funding support from HDRUK. S Gallier reports funding support from HDRUK, MRC and NIHR. E Sapey reports funding support from HDRUK, MRC, Wellcome Trust, NIHR, Alpha 1 Foundation, EPSRC and British Lung Foundation. A. Topham report no conflicts of interest.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

References

1 Goldacre B. *Better, Broader, Safer: Using health data for research and analysis. A review commissioned by the Secretary of State for Health and Social Care. Department of Health and Social Care*. 2022.

2 Department of Health & Social Care. Policy Paper. Data access policy update In: Care. DoHS, (ed.). 2023.

3 UK Data Service. What is the Five Safes framework?, https://ukdataservice.ac.uk/help/secure-lab/what-is-the-five-safes-framework/ (2024, accessed 22/05/2024).

4 Atkin C, Crosby B, Dunn K, et al. Perceptions of anonymised data use and awareness of the NHS data opt-out amongst patients, carers and healthcare staff. *Res Involv Engagem* 2021; 7: 40. 20210614. DOI: 10.1186/s40900-021-00281-2.

5 Gallier S, Price G, Pandya H, et al. Infrastructure and operating processes of PIONEER, the HDR-UK Data Hub in Acute Care and the workings of the Data Trust Committee: a protocol paper. *BMJ Health Care Inform* 2021; 28 2021/04/15. DOI: 10.1136/bmjhci-2020-100294.

6 Boyd A, Flaig R, Oakley J, et al. The UK Longitudinal Linkage Collaboration: A trusted research environment for the longitudinal research community. *International Journal of Population Data Science* 2023; 8.

7 Wilde K, Anderson L, Boyle M, et al. Introducing a new Trusted Research Environment - the Safe Haven Artificial Platform (SHAIP). *Int J Popul Data Sci* 2022; 7 20220825. DOI: 10.23889/ijpds.v7i3.2056.

8 Lyons RA, Jones KH, John G, et al. The SAIL databank: linking multiple health and social care datasets. *BMC Medical Informatics and Decision Making* 2009; 9: 3. DOI: 10.1186/1472-6947-9-3.

9 NHS England. Accessing data for research and analysis, https://transform.england.nhs.uk/key-tools-and-info/data-saves-lives/secure-data-environments/accessing-data-for-research-and-analysis/#:~:text=The%20NHS%20Research%20SDE%20Network%20is%20an%20interoperable%20England%2Dwide,ensure%20consistent%20standards%20and%20interoperability. (accessed 22/09/2024).

10 Kavianpour S, Sutherland J, Mansouri-Benssassi E, et al. Next-Generation Capabilities in Trusted Research Environments: Interview Study. *J Med Internet Res* 2022; 24: e33720. 20220920. DOI: 10.2196/33720.

11 Jefferson E, Liley J, Malone M, et al. GRAIMATTER green paper: Recommendations for disclosure control of trained machine learning (ML) models from trusted research environments (TREs). *arXiv preprint arXiv:221101656* 2022.

12 Veale M, Binns R and Edwards L. Algorithms that remember: model inversion attacks and data protection law. *Philos Trans A Math Phys Eng Sci* 2018; 376 20181015. DOI: 10.1098/rsta.2018.0083.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

13 Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods* 2020; 17: 261-272. 20200203. DOI: 10.1038/s41592-019-0686-2.

14 Harris CR, Millman KJ, van der Walt SJ, et al. Array programming with NumPy. *Nature* 2020; 585: 357-362. DOI: 10.1038/s41586-020-2649-2.

15 Office for National Statistics. National identify variable: Census 2021, https://www.ons.gov.uk/census/census2021dictionary/variablesbytopic/ethnicgroupnationalidentitylanguageandreligionvariablescensus2021/nationalidentity (2023, accessed 22/09/2024).

16 Information Commissioner's Office. Guidance on AI and data protection, https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/artificial-intelligence/guidance-on-ai-and-data-protection/how-should-we-assess-security-and-data-minimisation-in-ai/ (2023, accessed 22/09/204).

17 International Organization for Standardization. ISO/IEC 29500-1:2016. Information technology — Document description and processing languages — Office Open XML File Formats, https://www.iso.org/standard/71691.html (2022, accessed 22/09/2024).

18 Jefferson E, Cole C, Boixader AC, et al. GRAIMatter: Guidelines and Resources for AI Model Access from TrusTEd Research environments. In: *International Population Data Linkage Conference 2022* 2022.

19 Rolan G, Dalins J and Wilson C. The Data Airlock: infrastructure for restricted data informatics. *arXiv preprint arXiv:220309006* 2022.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Vital sign	Distribution	Population	Parameter	Value
Blood O_2 Saturation	Beta	1	alpha	5
			beta	0·25
		2	alpha	3
			beta	0·25
Systolic Blood Pressure	Normal	1	mean	128
			standard deviation	5
		2	mean	105
			standard deviation	5
Pulse Rate	Normal	1	mean	70
			standard deviation	15
		2	mean	70
			standard deviation	15
Temperature	Normal	1	mean	37
			standard deviation	0·5
		2	mean	38
			standard deviation	0·5
Respiratory Rate	Normal	1	mean	16
			standard deviation	2
		2	mean	19
			standard deviation	2
Supplementary Oxygen	Random Choice	1	oxygen probability	0·1
			room-air probability	0·9
		2	oxygen probability	0·2
			room-air probability	0·8
Consciousness (ACVPU)	Random Choice	1	alert probability	0·85
			CVPU probability	0·15
		2	alert probability	0·78
			CVPU probability	0·22
Sex	Random Choice	1	male probability	0·57
			female probability	0·43
		2	male probability	0·43
			female probability	0·57
Ethnicity	Random Choice	equal probabilities across all values		
Age	Random Choice	equal probabilities across all values		

Table 1: Parameters used to generate random data for file samples presented herein. Pulse rate was selected to be an uninformative variable, and so parameters were the same for both populations. Age values were all whole integers ranging from 18 to 95. Ethnic groups were selected by equal probability, samples from the Office for National Statistics high-level ethnic categories for the 2021 census.[15]

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Property	Type
decision_function_shape	str
break_ties	bool
kernel	str
degree	int
gamma	str
coef0	float
tol	float
C	float
nu	float
epsilon	float
shrinking	bool
probability	bool
cache_size	int
class_weight	NoneType
verbose	bool
max_iter	int
random_state	NoneType
_sparse	bool
n_features_in_	int
class_weight_	numpy.ndarray
classes_	numpy.ndarray
_gamma	numpy.float64
support_	numpy.ndarray
_n_support	numpy.ndarray
dual_coef_	numpy.ndarray
intercept_	numpy.ndarray
_probA	numpy.ndarray
_probB	numpy.ndarray
fit_status_	int
_num_iter	numpy.ndarray
shape_fit_	tuple
intercept	numpy.ndarray
_dual_coef_	numpy.ndarray
n_iter_	numpy.ndarray

Table 2: Instance properties of a support vector machine classifier trained using the popular SciKit-Learn Python package ("SVC" class defined in the sklearn.svm namespace). Highlighted in red is the property that contained a large subset of the patient-level training data identified in Case three. Note also that in Python any one of the properties of a class object can be assigned at-will by external code, and so it is trivial to assign any arbitrary numpy.ndarray object (a data matrix of fixed type but arbitrary dimensions) to any of the relevant properties of a class that normally contains them – the name and normal function of the property are absolutely not reliable indicators as to the content of the property in any given instance when egressing data.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

Case	Description	Data Involved	Format	Challenges	Insights	Guidelines
Case one **(High Risk)**	Unintentional inclusion of performance data from AI models in binary-serialised files.	Individual level data on infections, patient demographics, diagnoses, comorbidities, lab results.	Binary object serialisation (Python's pickle method).	Lack of awareness of permissible exports; complexity of verification process for pickled files.	Tools like object serialisation can easily include sensitive data; thorough manual inspection required for verification.	Educate researchers on permissible exports; implement rigorous verification processes for pickled files.
Case two **(Critical Risk)**	Unintentional embedding of a Support Vector Machine with training data in a complex object structure that was binary-serialised.	AI models for clinical decision support, patient-level training data.	Binary object serialisation (Python's pickle method).	Complexity of object structure; difficulty in identifying and verifying embedded data.	Complexity of object structure can conceal sensitive data; expertise of platform and platform-independent risks posed by different AI models required for thorough verification.	Develop expertise in platform used and knowledge of AI models; implement recursive inspection processes for complex objects.
Case three **(High Risk)**	Export of aggregated results as a complex and large set of markup language with embedded files.	Aggregate data without low-number suppression, arbitrary contents of embedded files.	HTML	Complexity of object structure, technical demands on verification.	Plain-text markup documents can easily be spiked by malicious users and require technicians with good expertise to verify their contents. Requires implicit trust in library creators that users cannot accidentally embed sensitive data.	Thoroughly inspect code and image metadata for sensitive data; ensure expertise in verification process, encourage use of plain-text aggregate data rather than formatted reports.

Table 3: Summary of all cases discussed herein, including the description, data involved, method of export, challenges faced, insights gained, and guidelines for practitioners to mitigate similar risks.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

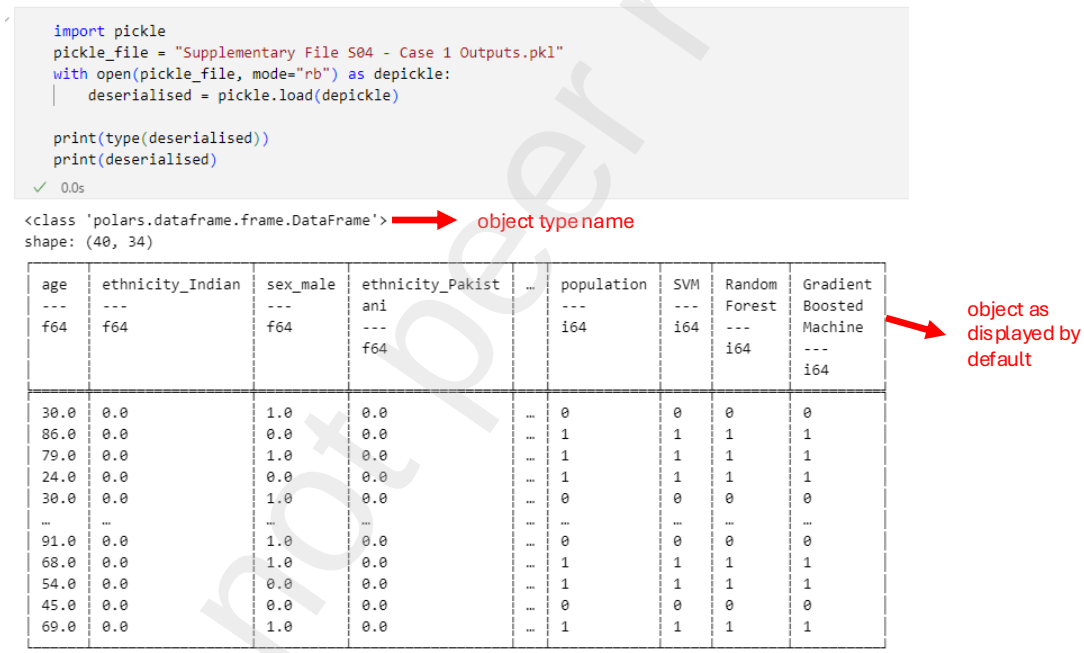

Figure 1: Case 1 as viewed in VS Code. Screenshot taken from VS Code of Case 1 example output file, showing the object type name and the contents of the object highlighted in red. In this instance, the default display rendering of the object presented the data in a clear and very readable way, and the object type immediately reveals the nature of the object if not the content.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

```
# yes, it's an SVC object, let's just check its attributes
# according to the docs, it should have a property called "support_vectors_"
properties = dir(deserialised["svm"]["model"])
properties.reverse()
print(properties)
```

✓ 0.0s

```
['verbose', 'unused_param', 'tol', 'support_vectors_', 'support_', 'shrinking', 'shape_fit_', 'set_
```

suspect property confirmed

```
# ... confirmed: it's there, let's just check them out
import polars as pl

svs = deserialised["svm"]["model"].support_vectors_
suspect = pl.DataFrame(svs)
print(suspect)
suspect.write_excel("Supplementary File S05c - Case 2 Support Vectors.xlsx")
```

✓ 0.3s

"support_vectors_" property formatted for readability

```
shape: (103, 30)
```

column_0	column_1	column_2	column_3	…	column_26	column_27	column_28	column_29
f64	f64	f64	f64		f64	f64	f64	f64
39.0	0.0	1.0	0.0	…	0.0	0.0	119.0	0.0
79.0	0.0	1.0	0.0	…	0.0	0.0	128.0	0.0
32.0	0.0	1.0	0.0	…	1.0	0.0	130.0	0.0
84.0	0.0	0.0	0.0	…	0.0	0.0	126.0	1.0
72.0	0.0	1.0	0.0	…	0.0	0.0	125.0	0.0
…	…	…	…	…	…	…	…	…
82.0	0.0	1.0	0.0	…	0.0	0.0	106.0	0.0
62.0	0.0	1.0	0.0	…	0.0	0.0	104.0	0.0
66.0	0.0	0.0	0.0	…	1.0	0.0	112.0	1.0
41.0	0.0	1.0	0.0	…	0.0	1.0	107.0	0.0
82.0	0.0	0.0	0.0	…	0.0	0.0	104.0	1.0

Figure 2: Case 2 as viewed in VS Code. Screenshot of VS Code showing fragment of **Supplementary Code file S05b** wherein the presence of support vectors was confirmed for a support vector machine classifier object embedded in **Online Supplementary file S05a**. Highlighted in red are the points where the suspect property's presence in the object was confirmed, and a rendering of the property's contents.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

A

```
                "temperature": temp_pop2,
                "ethnicity": ethnicity_pop2,
                "age": age_pop2,
                "sex": sex_pop2
            })
        ],
        how="vertical"
    )
)

# print(dataset)
```

```
dataset.write_excel("experiment data.xlsx")
```

```
<xlsxwriter.workbook.Workbook at 0x1ac7f678430>
```

B


```
  {
   "data": {
    "text/plain": [
     "<xlsxwriter.workbook.Workbook at 0x1ac7f678430>"
    ]
   },
   "execution_count": 23,
   "metadata": {},
   "output_type": "execute_result"
  }
 ],
 "source": [
  "dataset.write_excel(\"experiment data.xlsx\")"
 ]
}
],
"metadata": {
 "kernelspec": {
  "display_name": ".venv",
  "language": "python",
  "name": "python3"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.10.5"
 }
},
"nbformat": 4,
"nbformat_minor": 2,
"patients": [
    {
        "sex": "female",
        "age": 38,
```


insertion of patient data

Figure 3: Patient data embedded in a Jupyter Notebook. Partial screenshots of the code markup file in supplementary file S03. A shows a screenshot of the end of the file as rendered in VS Code at the location into which the synthetic patient data was inserted, and B shows the markup for the same screen area displayed in a plain text editor (Notepad++), marked in red is the site in which synthetic patient data has been inserted.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496

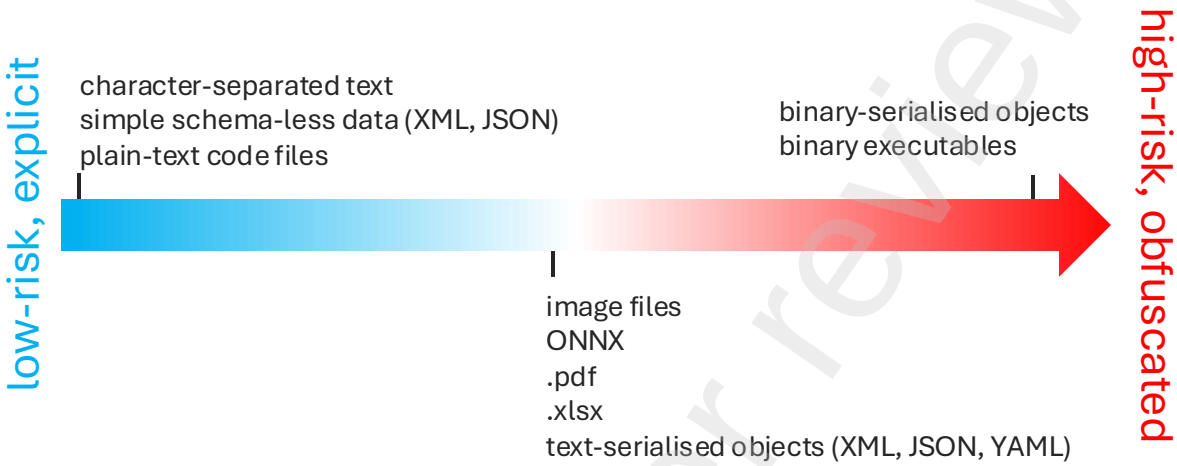

Figure 4: Spectrum of risk for files egressed from the PIONEER TRE, the scale of which depends on the type of file proposed for egress. Plain-text is fully human-readable, but depending on the way it is used can be relatively complex in nature, but always explicit. Non-human-readable formats, typically binary objects derived from arbitrary, user-defined types present the greatest risk.

Preprint not peer reviewed

This preprint research paper has not been peer reviewed. Electronic copy available at: https://ssrn.com/abstract=5001496